TÜBİTAK BİLGEM YTE | YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

KİMLİK DOĞRULAMA ALTYAPILARININ KULLANIMI

SAYI: 04

ARAŞTIRMA SERİSİ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
SSO	Single Sign-On (Tekil Oturum Açma)
CAS	Central Authentication Service (Merkezi Kimlik Doğrulama Servisi)
TGT	Ticket Granting Ticket (Bilet Veren Bilet)
ST	Service Ticket (Servis Bileti)
SAML	Security Assertion Markup Language (Güvenlik Onaylama İşlemi Biçimlendirme Dili)
OIDC	OpenID Connect
OAuth	Open Authorization (Açık Yetkilendirme)
JWT	JSON Web Token
HTTPS	Hypertext Transfer Protocol Secure (Güvenli Metin Aktarım İletişim Protokolü)
UI	User Interface (Kullanıcı Arayüzü)
CSS	Cascading Style Sheets (Basamaklı Stil Şablonları)
FTL	Freemaker Template (Freemaker Şablonu)
SPI	Service Provider Interface (Servis Sağlayıcı Arayüz)

Yazar
Deniz GÜRER

Yayın Koordinatörü
Elif ŞENYİĞİT

Editörler
Muhammed Fatih DOĞMUŞ
Sevinç KARAKAŞ
Tuğçe YILMAZ

Tasarım
Şeyma Koçer

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

yte.bilgem.tubitak.gov.tr

İçindekiler

Önsöz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü, 2012 yılından bu yana yazılım teknolojilerinde AR-GE faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

Araştırma Serisi ile TÜBİTAK BİLGEM YTE kurum içi çalışmaların yaygınlaştırılması ve sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanması hedeflenmektedir.

Giriş

Kullanıcı kimlik doğrulaması ve yetkilendirilmesi için pek çok teknoloji mevcuttur. Bunlar arasında Apereo CAS, Keycloak, Gluu, Okta, Shibboleth, WSO2, AWS Cognito sayılabilir. Bu çalışma kapsamında açık kaynak olmaları, kapsamları ve ihtiyaçlara göre genişletilebilirlikleri açısından öne çıkan web tabanlı uygulamalara güvenli tekil oturum açma (Single Sign On / SSO) hizmetine sahip Apereo CAS ve Keycloak teknolojileri incelenecektir.

Bu çalışmada, yazılım projelerinin önemli bir parçası olan kullanıcı girişi ve yetkilendirilmesi teknolojileri hakkında alternatifler araştırılmış ve Spring Framework ile entegre edilebilirliği test edilmiştir. İlgili teknolojilerin günümüz ihtiyaçlarına cevap verilebilirliği gözlemlenmiş, birbirlerine göre avantajları ve dezavantajları karşılaştırılmış ve mevcut projelere ya da gelecek geliştirilebilecek projelere uygulanabilirliği yorumlanmıştır. Çalışma, yazılım projelerine güvenlik katmanı eklemek isteyen her kamu kurumu ya da özel sektör için kullanılabilecek kapsamda ele alınmıştır.

Günümüzde bir SSO (Single Sign On) teknolojisinin hangi hizmetleri sağlaması gerektiğine dair detaylı araştırma yapılmış, örneklendirilmiş ve dokümante edilmiştir.

Apereo CAS

CAS (Central Authentication Service), web tabanlı uygulamalara güvenli tekil oturum açma (Single Sign On / SSO) hizmeti sağlayan, açık kaynak kodlu merkezi kimlik denetimi yazılımıdır. CAS, 2000 yılında Yale Üniversitesi'nde geliştirilmeye başlanmış, 2004 yılında ise Jasig'e (Java in Administration Special Interest Group) devredilmiştir. 2012 yılında Jasig, Sakai Vakfı ile birleşerek Apereo adını almıştır. CAS projesi, günümüzde Apereo Vakfı tarafından desteklenmektedir. Apereo CAS, merkezi tekil oturum açma, kullanım kolaylığı ve güvenlik avantajları sağlar. Tek bir kullanıcı adı ve parola girerek oturum açma ile birden fazla servise erişim mümkün hale gelirken, her servisin ayrı ayrı kullanıcı adı, parola gibi erişim bilgilerini işlemesi ortadan kalkmış ve bu şekilde güvenlik riskleri de azaltılmıştır. Örneğin; Google tarafından sağlanan herhangi bir hizmete erişilmek istendiğinde, bu ister mail.google.com ister drive.google.com olsun, her zaman accounts.google.com adresine yönlendirir. Bu accounts.google.com adresi, Google'ın kullanmış olduğu SSO hizmetidir. Burada giriş yapıldıktan sonra, bir başka Google servisine gidildiğinde tekrar o serviste giriş yapmaya ihtiyaç kalmaz.

1. CAS Sistem Bileşenleri

CAS sunucusu (CAS Server) ve istemcileri (CAS Clients), CAS sistem mimarisinin çeşitli protokollerle iletişim kuran iki fiziksel bileşenini oluşturur.

1.1. CAS Sunucusu

Temel sorumluluğu kullanıcıların kimliklerini doğrulamak ve yaygın olarak CAS istemcileri olarak adlandırılan CAS'ın etkin olduğu hizmetlere erişim sağlamak ve bilet (ticket) verip bu biletleri onaylamak olan Spring Framework ile inşa edilmiş Java Servlet'idir. Bir SSO oturumu, sunucu başarılı oturum açtıktan sonra kullanıcıya bir bilet (Ticket Granting Ticket: TGT) ulaştırdığında oluşturulur. TGT'yi bir token olarak kullanan tarayıcı, yönlendirme yoluyla kullanıcının isteğine göre bir servise, servis bileti (Service Ticket - ST) verir. Daha sonra CAS sunucusunda back-channel iletişimi ile ST'nin geçerliliği denetlenir (validate). CAS sunucusu, kimlik doğrulama işlemini farklı methodlarla yapacak şekilde yapılandırılabilir. CAS, pek çok yaygın kimlik doğrulama teknolojisi (LDAP, Database, JAAS, X.509 vs.) için gerekli bileşenleri sağlamaktadır. Bunun yanı sıra CAS sunucusu tarafından kullanılan arayüzler (AuthenticationHandler) genişletilerek, CAS'ın farklı kimlik doğrulama mekanizmaları kullanması da sağlanabilir.

1.2. CAS İstemcisi

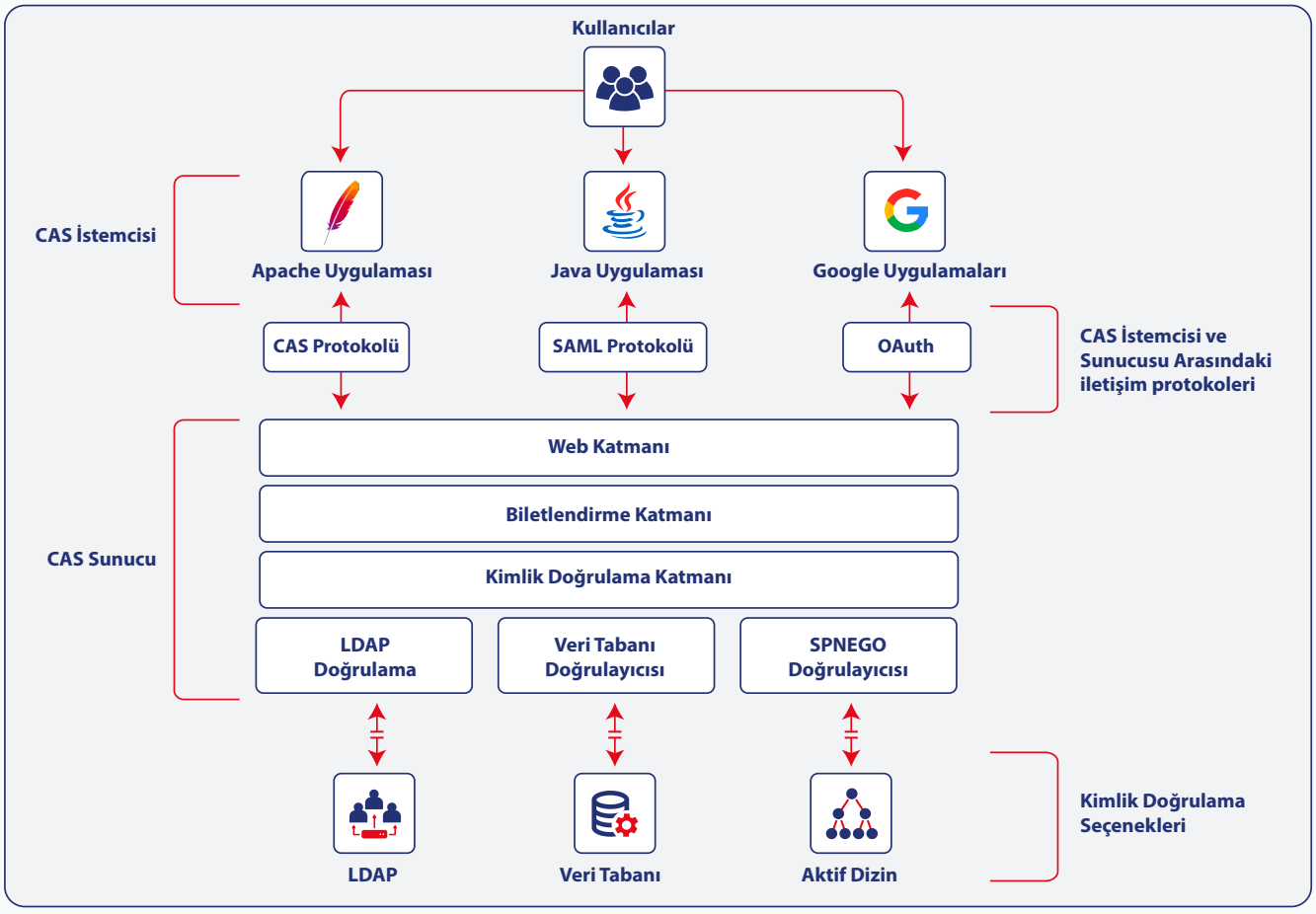
CAS İstemcisi, kimlik doğrulama (Authentication) işlemini CAS'a devreden uygulamalardır. Bu uygulamaların CAS Server ile iletişimini sağlayan istemci entegrasyon kütüphaneleri, farklı dil ve frameworkler için (Java, .NET, PHP, Spring Security, Apache Shiro) CAS tarafından hazırlanmış ve kullanıma sunulmuştur. CAS istemciler ile CAS sunucusu arasındaki iletişim, varsayılan olarak CAS protokolü ile yapılmaktadır. Ancak istenirse bu iletişimi farklı protokoller (OpenID, SAML, OAuth) kullanarak gerçekleştirmek de mümkündür. CAS yazılımının entegre olabildiği platformlar ve desteklediği protokoller aşağıda listelenmiştir:

CAS Sunucusunun Entegre Olabildiği Platformlar:

- Java (Java CAS Client)
- .NET (.NET CAS Client)
- PHP (phpCAS)
- Perl (PerlCAS)
- Python (pycas)
- Ruby (rubycas-client)

CAS Sunucusunun Desteklediği İletişim Protokolleri:

- CAS
- SAML 1.1 ve 2
- OpenID Connect
- OAuth 2.0
- WS Federation



Şekil 1. CAS Mimarisi

2. CAS Yazılım Bileşenleri

CAS sunucusu üç katmanlı alt sistemler kullanılarak tanımlanmaktadır:

- Web (Spring MVC/Spring Webflow)
- Biletlendirme (Ticketing)
- Kimlik denetimi (Authentication)

Web katmanı, CAS istemcileri arasında iletişim için uçlar (endpoints) sağlar ve sahip olduğu view'ları webflow sayesinde yönlendirilmiş uygulama (Guided Application) haline getirir. Kullanıcı, CAS üzerinde kimlik doğrulaması yaptıktan sonra, CAS tarafından SSO oturumu (session) yaratılır ve TGC isminde çerez (cookie) ile TGT (Ticket Granting Ticket) sağlanır. Bu TGT kullanılarak CAS istemcilerine ST (Service Ticket) sağlanır ve CAS'ta bu bilet (ticket) denetlenir. Bu işlemler, biletleme (ticketing) katmanında gerçekleştirilir. Bu işlemler için daha ayrıntılı akış aşağıda anlatılmıştır. Henüz CAS'a giriş yapmamış bir kullanıcı, CAS istemcisi uygulamaya istek attığında, giriş yaptırılması için HTTP 302 cevabı dönülür. Kullanıcı bu isteği aldığında CAS'a otomatik istek atar. Burada kullanıcı adı ve şifre girildikten sonra TGC isminde çerez ayarlanır ve kullanıcıya tek kullanımlık ST verilir. Kullanıcının CAS'a yönlendirilmesine neden olan uygulamaya tekrar gitmesi için HTTP 302 cevabı dönülür. Kullanıcı, bu ST'yi kullanarak uygulamaya istek atar. Uygulama gelen istekteki ST'yi denetlemek için CAS'a istek atar, Eğer doğrulama sağlanırsa HTTP 200 dönülür ve uygulama da, kullanıcıya giriş yapıldığını belirten oturum bilgisini JSESSIONID çerezi ile cevap döner. Kullanıcı, bundan sonraki tüm isteklerinde bu JSESSIONID'yı ekler ve CAS'a gerek kalmadan uygulama üzerinden oturum doğrulama gerçekleşir. Kullanıcı başka bir CAS istemcisi uygulamaya erişim sağladığında, burada giriş yapmamış olduğundan giriş yapması için kullanıcıya HTTP 302 cevabı dönülür. Kullanıcı, bu isteği aldığında CAS'a otomatik istek atar. Gelen istek içerisinde TGC olduğundan kullanıcı adı-şifre işlemi atlanır. Kullanıcıya direkt tek kullanımlık ST verilir ve akış yukarıdaki gibi devam eder.

3. Kurulum

CAS, kodu indirip derlemek yerine zaten derlenmiş haline eklemeler/çıkarmalar yapmayı sağlamak için overlay adında şablon (template) sağlar. Yani tüm kodları indirip, içerisinde değişiklik yaparken kaybolmak yerine bu sade hali üzerinde kolayca değişiklik yapılabilir. Overlay template ile binary dosyalar indirilir, ekleme/çıkarma ile yapılandırma dosyaları binary hale getirilip eskileri ile değiştirilir ve son olarak kalan binary dosyaları ile birleştirilip war altında arşiv oluşturulur. Java sınıflarını, resources ve static dosyalarını (js, css, images) değiştirebilmek için src klasörü altında yapılandırma dosyaları oluşturulur. Bunlar, önceden indirilmiş binary dosyaları ile paket konumu ve ad olarak tam eşleşmelidir. src klasörü dışında yapılan tüm değişiklikler derleme aşamasında kaybolur. Sistemde gradle yüklü olmalıdır ve JAVA_HOME, Java yolunu (path) göstermelidir.

```
Kurulum Kaynak Kodu
https://github.com/apereo/cas-overlay-template.git
```

4. Paket Yapısı

- /etc/cas/config altındaki log4j2.xml ile uygulama log ayarları ve cas.properties ile uygulama ayarları yapılır.
- /etc/cas/services ile cas istemcilerinin kaydı yapılır.
- /etc/cas/thekeystore ile SSL sertifika kaydı yapılır.
- /src altında, yeni eklenmek ya da ezilmek istenen yapılandırmalar bulunur.

Yeni oluşturulan sınıflarda, CAS'ta olmayan bağımlılıkları eklemek için /build.gradle dosyası kullanılır. Yukarıda bahsedilen klasör ya da dosyalarda herhangi değişiklik yaptıktan sonra overlay'i kullanabilmek için aşağıdaki komutlar çalıştırılır.

```
gradlew.bat copyCasConfiguration
gradlew.bat explodeWar
gradlew.bat run
```

Loglarda READY yazısı görüldükten sonra http://localhost:8443/cas/login üzerinden CAS giriş ekranına giriş yapılır.

Bir diğer çalıştırma yöntemi de docker kullanmaktır.

```
gradlew.bat jibDockerBuild
docker run --name cas -p 8443:8443 -d org.apereo.cas/cas
docker logs -f cas
```

5. CAS İstemcilerinin Kayıt Edilmesi

CAS sunucusunu kullanarak merkezi kimlik denetimi yapmak isteyen uygulamaların, önceden CAS sunucusu tarafından biliniyor olması gerekir. Örneğin; https://auth.berkeley.edu/cas/login'e istek atıldığında, "uygulama, merkezi kimliklendirme servisini kullanmak için yetkilendirilmemiş" hatası ile karşılaşılır. İstek, CAS'a kaydedilmiş bir istemci üzerinden yenilendiğinde giriş ekranı gelir.
(https://auth.berkeley.edu/cas/login?service=http://www.berkeley.edu)

CAS sunucusu, merkezi kimliğe tabi olacak uygulamaların listesini tutmak için 2 farklı yapı sunar:

- Statik Yöntem: CAS Sunucusunu kullanacak istemci uygulamaları, CAS Sunucusunun tanım dosyalarına (/etc/cas/services altında json/yaml dosyaları) hepsinin benzersiz bir ID numarası olacak şekilde eklenir, derleme işlemi yapılır ve CAS sunucusunun yeni sürümü alınarak kurulumu yapılır. Yeni bir CAS istemcisi ekleneceği zaman bu işlem tekrar edilir. Örnek olarak, geliştirme ortamında 8094 portunda çalışan uygulamanın CAS'ı kullanabilmesi için gereken servis tanım dosyası şu şekildedir:

```
{
 “@class” : “org.apereo.cas.services.RegexRegisteredService”,
“serviceId” : “http://localhost:8094/.*”,
“name” : “CAS Spring Secured App”,
“description”: “This is a Spring App that uses the CAS Server for it’s
authentication”, “id” : 19991, “evaluationOrder” : 1
}
```

- Dinamik Yöntem: CAS Management Application uygulamasının web arayüzünden CAS istemci uygulamalarının URL’leri dinamik olarak tanımlanabilir. CAS istemci değişikliği durumunda, derleme ve yeniden sürüm kurulumu gerektirmez.

6. Varsayılan Ekranları Güncelleme

```
gradlew.bat listTemplateViews
> Task :listTemplateViews \cas\build\cas-resources\templates\
casAcceptableUsagePolicyView.html \cas\build\cas-resources\templates\
casAccepttoRegistrationView.html \cas\build\cas-resources\templates\
casAccountDisabledView.html \cas\build\cas-resources\templates\
casAccountLockedView.html \cas\build\cas-resources\templates\casAdminLoginView.
html
```

Yukarıdaki komut, CAS’ın sahip olduğu tüm varsayılan ekranları ve dosya yollarını gösterir. Bu ekranlarda herhangi bir güncelleme yapmak için, hangi ekran güncellenecek ise src/main/resources/templates altında aynı adla boş bir view oluşturulur. Varsayılan ekran içeriği, bu yeni dosyaya kopyalanır ve istenilen değişiklikler yapılır. Değişikliklerin yansıması için tekrar derlenip CAS’ı ayağa kaldırmak gereklidir. Eğer, üstteki adımlarda komut çıktısı ve build/cas-resources/templates altı boş ise, CAS’ın asıl kod tabanı (codebase) üzerinden varsayılan ekranlar bulunabilir. (modül ismi: cas-server-support-thymeleaf)

7. Özel Ekran Teması Geliştirme

Temalar aracılığıyla şablonlar (templates) oluşturulur ve varsayılan CSS, JS ya da resim dosyaları yerine, belirlenen tema içindeki dosyalar kullanılır. Temalar ile CAS’ı kullanan her istemci için ayrı bir görünüm ve davranış elde edilebilir.

CAS’ta UI kısımları 3 şekilde ele alınır. Bu kısımlar, cas-server-support-thymeleaf modülünden otomatik olarak yüklenir ve CAS tarafından kullanılır.

- views: HTML dosyaları (classpath:/templates dizini altında)

- themes: view dosyalarının kullanıldığı CSS, JS ve resim dosyaları (classpath:/static altında css,js,images alt klasörleri)
- message bundles: view'larda gösterilecek tüm metinleri tanımlayan dosyalar (classpath:/ altında)

Varsayılan tema, classpath:/cas-theme-default.properties tarafından tanımlanır.

```
cas.standard.css.file=/css/cas.css
cas.standard.js.file=/js/cas.js
```

CAS'ta UI düzenlemek için 2 yaklaşım vardır.

- Dekoratif (decorative) bileşenler (CSS, JS) düzenlenir, yapısal (structural) bileşenler (HTML views) aynı kalır.
- İkisi birden düzenlenir.

İlki, yeni tema yaratmak demektir (diğer bir deyişle varsayılan view'ların görünümünü yeniden şekillendirme). Yeni tema icin classpath'de {theme-name}.properties olmalı, static altında theme-name adında bir klasör olmalı ve css, js, images alt klasörleri içermelidir.

```
cas.standard.css.file=/themes/{theme-name}/css/cas.css
cas.standard.js.file=/themes/{theme-name}/js/cas.js
```

Tema oluşturulduktan sonra cas.properties'te ya da cas istemcilerinin tanımının yapıldığı service registry'de, tema tanımı yapılmalıdır.

```
cas.theme.defaultThemeName={theme-name}

ya da

“theme” : “[theme_name]”
```

Tema oluşturulduktan sonra cas.properties'te ya da cas istemcilerinin tanımının yapıldığı service registry'de, tema tanımı yapılmalıdır.

8. CAS Kimlik Doğrulama Yöntemleri

CAS; veritabanı, REST, SOAP gibi birçok kimlik doğrulama methodu (https://apereo.github.io/cas/6.1.x/installation/Configuring-Authentication-Components.html) sunmaktadır. Seçilen methodu kullanabilmek için, önce bağımlılığın yüklenmesi/eklenmesi, ardında da cas.properties kısmında yapılandırmaların yapılması gereklidir. Örneğin, REST kimlik doğrulamasını kullanabilmek için org.apereo.cas:cas-server-support-rest-authentication modülü eklendikten sonra cas.properties kısmında şu ayarlamalar yapılır.

```
cas.authn.rest.uri=http://localhost:8082/validate
cas.authn.rest.passwordEncoder.type=NONE
```

CAS, giriş ekranında girilen kullanıcı adı ve şifre ile Basic Authorization Header oluşturup, rest.uri ile belirtilen güvenlikli adrese istek atar. Eğer Basic Authorization doğrulanırsa, içerisinde kullanıcı adının olduğu 200 cevabı döner ve CAS tarafında da doğrulama yapılmış olur.

9. Canlı (Production) Ortama Hazırlık

Bu bölüm, canlı ortama hazır CAS ortamı için gereken genel yapılandırma seçeneklerini açıklamaktadır. Buradaki bilgiler, genel kavramlar ve gerçek uygulamaları içerir.

9.1. Güvenli Bağlantı (HTTPS) Oluşturma

Geliştirme ortamında SSL olmadan HTTP üzerinden işlemler yapılmak istendiğinde, TGC çerezi tarayıcıda kayıt edilemez. Bu durumda CAS ekranında giriş yaptıktan sonra, tekrar CAS ekranına gelindiğinde, sanki önceden giriş yapılmamış gibi tekrar giriş ekranı gösterilir. Halbuki bu durumda "Siz daha önceden giriş yaptınız, bu da giriş bilgileriniz." şeklinde bir sayfa gösterilmesi beklenir. Bunun nedeni, TGC çerezini ayarlayan cevabın (response) başlığında (header), bu işlemin yalnızca SSL bağlantıları üzerinden gerçekleşmesi gerektiğinin belirtilmiş olmasıdır (varsayılan olarak cas.tgc.secure=true ayarlı). Dolayısıyla, web tarayıcısı bu çerez ayarlama işlemini engelleyecektir.

Tarayıcı loglarında şu mesaj gözlemlenir:

```
the set cookie was blocked because it had the "secure" attribute but was not
received over a secure connection.
```

Bu yüzden, geliştirme ortamında HTTP üzerinden işlemler yapılırken cas.tgc.secure=false olarak ayarlanmalıdır. Fakat canlı (production) ortamda çalışan uygulamanın güvenli bağlantı üzerinden çalışması, güvenlik risklerinin önüne geçmektedir. Bu yüzden HTTPS bağlantısı ile çalışacak şekilde ayarlanmalıdır. Sertifika oluşturma aşamasında firstname ve lastname alanlarına sunucu IP ya da DNS adresi girilmelidir. Yoksa SSL handshake sırasında hata ile karşılaşılır.

```
keytool -genkey -keyalg RSA -alias thekeystore -keystore thekeystore
-storepass changeit -validity 360 -keysize 2048 -ext san=ip:10.222.130.x (yada
san=dns:<dns-name>)
```

Oluşturulan bu sertifikanın istemci uygulamaları tarafından kullanılabilmesi için, .der formatında export edilmesi gerekir. Yalnızca bu formatta ise kabul edilir.

```
keytool -export -alias thekeystore -file thekeystore.der -keystore thekeystore
keytool -import -alias thekeystore -storepass changeit -file thekeystore.der
-keystore $JAVA_HOME\lib\security\cacerts
```

9.2. TGC İçeriği

CAS sunucusu, SSO oturumu boyunca giriş durumunu sağlamak için TGT'yi kullanır. Redis'te bu değer TGT-1-xyz-denizg-l tarzındayken, TGC çerezi içerisinde JWT olarak tutulmaktadır. Bunun nedeni, CAS sunucusu TGT'yi tarayıcıya göndermeden önce, güvenliğini sağlamak için JWT oluşturarak, TGT'yi JWT payload içerisine koyar, daha sonra imzalar, daha sonra da JWT payload'ını şifreler. Eğer bu şifreleme anahtarı ve imzalama anahtarı tanımlanmazsa, CAS uygulama ayağa kalkma aşamasında kendi otomatik olarak anahtarlar oluşturur.

```
WARN [ ... ] - Secret key for encryption is not defined. CAS will auto-generate
the encryption key
WARN [ ... ] - Generated encryption key ABC of size  ... The generated key MUST be
added to CAS settings
```

Birçok sunuculu ortamda çalışırken, tüm sunucular aynı anahtarları kullanmalıdır, aksi takdirde tanımlanmazsa her sunucu kendi şifreleme/imzalama anahtar ikilisini üretecek ve her sunucu birbirinden farklı anahtarları kullanacaktır. Bu durumda, bir sunucuda üretilen TGT, diğer sunucuya geldiğinde kullanılamayacak ve dahası, var olan TGT silinerek yerine boş bir TGT gönderilecektir.

```
cas.tgc.crypto.signing.key=dasdasd
cas.tgc.crypto.encryption.key=dasdad
cas.tgc.crypto.enabled=true
```

9.3. CAS Bilet Kaydı (Ticket Registry)

Biletler (tickets), Spring Data Redis yardımıyla Redis'te saklanır. Bu sayede, çok sunuculu ortamda, her bir sunucu, aynı bilet üzerinde işlem yapabilir. Bu biletler için anahtar isimleri, CAS:Ticket ile başlar.

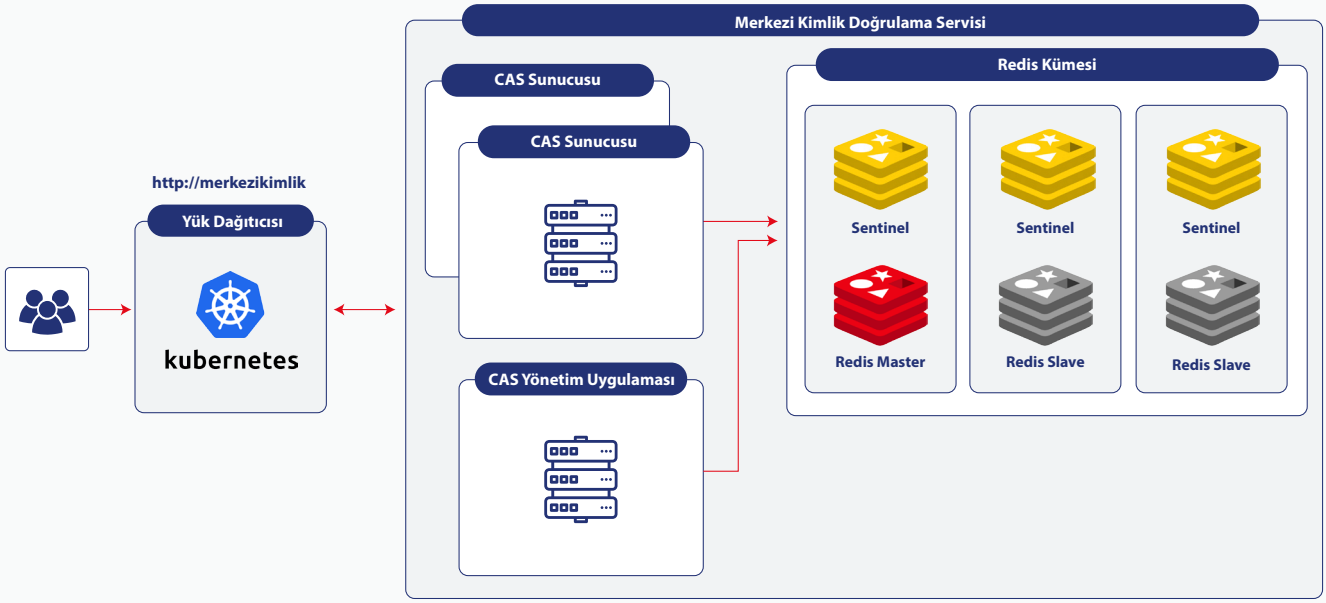
```
cas.ticket.registry.redis.sentinel.master=redis-cluster cas.ticket.registry.
redis.sentinel.node[0]=10.x.y.z:26379 cas.ticket.registry.redis.sentinel.
node[1]=10.x.y.z:26379 cas.ticket.registry.redis.sentinel.node[2]=10.x.y.z:26379
cas.ticket.registry.redis.password=test-sifre
```

9.4. Yüksek Erişilebilirlik

Yüksek erişilebilirliğin (high availability) sağlanması için CAS'ın küme (clustered) biçimde tasarlanması önerilmektedir.

Yüksek erişilebilirliği sağlamak için, Şekil 2'de görülen mimari tasarım örnektir. Birçok CAS sunucusu, Kubernetes tarafından yönetilebilir şekilde konfigüre edilebilir. Bu konfigürasyon ile CAS sunucu uygulamalarından herhangi birisinin çöktüğü durumda, Kubernetes'in küme orkestratörü tarafından başka bir sunucu üzerinde kaldırılmaktadır. CAS Sunucularının bir hata durumunda servisler için oluşturdukları biletlerin

(TGT, ST) kaybedilmemesi için Redis, master-slave yapısında replike edilmiştir. CAS Sunucu uygulamaları, yük dengeleyicisi (load balancer) arkasından sunulmaktadır.



Şekil 2. CAS'ın Örnek Küme Yapısı

10. Spring Framework Entegrasyonu

Spring Security'nin CAS ile entegre edilebilmesi için, CAS istemcisi olduğunu belirten bir kütüphane eklemelidir. (org.springframework.security:spring-security-cas)

Kullanıcı uygulamaya ilk defa istek attığında AuthenticationException ile karşılaşılır. Bu noktada, casAuthenticationEntryPoint ile CAS adresi tanımlayarak CAS'a yönlendirme yapılması gereklidir. İkinci olarak, CAS'a giriş yapıldıktan sonra ST ile uygulamaya gelen istekteki bileti doğrulamak için yapılandırmalar gereklidir. UsernamePasswordAuthenticationToken ile ST'yi temsil edecek sınıf oluşturulur. Uygulamaya ST içeren istek geldiğinde (bu da UsernamePasswordAuthenticationToken), yalnızca bu tip isteklere cevap verecek casAuthenticationProvider oluşturulur ve ticket'ı doğrulamak için TicketValidator arayüzünü implement eden Cas20ServiceTicketValidator nesnesi oluşturulur. Bu sınıf, CAS'a ST'yi doğrulamak için istek atar. CAS da, içerisinde kullanıcı adı olan bir XML cevap döner. Cas20TicketValidator, dönen XML'i parse eder ve kullanıcı adını içeren TicketResponse oluşturur. casAuthenticationProvider, bu kullanıcının sahip olduğu rolleri (GrantedAuthority) yüklemek için AuthenticationUserDetails'e istek atar ve en sonda sahip olduğu TicketResponse ve GrantedAuthority bilgileri ile casAuthenticationToken oluşturur. Bu noktada kontrol tekrar casAuthenticationFilter sınıfına geçer ve casAuthenticationToken'ı SecurityContext'e yerleştirilir. AuthenticationException'a neden olan sayfaya geri yönlendirme yapılır.

Keycloak

Keycloak, web tabanlı uygulamalara güvenli tekil oturum açma (Single Sign On / SSO) hizmeti sağlayan açık kaynak yazılımdır. Keycloak, 2013 yılında geliştirilmeye başlanmış ve ilk sürümü Eylül 2014'te yayınlanmıştır. 2016 yılında Red Hat, SSO ürününü olan PicketLink projesini Keycloak ile birleştirmiştir. Keycloak, eskiden Wildfly uygulama sunucusu üzerine kurulu bir uygulama iken, Keycloak 2017 itibariyle Quarkus framework'ü

üzerine inşa edilmis bir uygulama oldu. Wildfly versiyonu, konfigürasyon için karmaşık XML dosyaları barındırıyordu. Bu da sürüm yükseltme aşamasında hatalara yol açıyordu.

Keycloak şu özellikleri sağlar:

- Kimlik Doğrulama
- Kullanıcı Yetkilendirme
- Kullanıcı Yönetimi
- User Federation (Kimlik bilgilerinin Keycloak harici yerlerde saklanması ve bu credentialları görmeden kimlik kontrolünün SAML, OAuth, OpenID Connect gibi yöntemlerle yapılmasıdır.)

1. Tenant ve Realm

Yazılım terminolojisinde tenant, bir organizasyona hizmet eden uygulama demektir. Multi-tenant ise farklı organizasyonlara hizmet eden uygulama demektir. Keycloak'ta tenant'ın karşılığı olarak realm terimi kullanılır. Birbirinden izole edilmiş kullanıcı ve uygulama grupları oluşturmaya izin verir. Realm, güvenlik alanı (security domain) olarak görev yapar. Keycloak, varsayılan olarak master isminde tek bir realm ile gelir. Bu, Keycloak'ı yönetmek için kullanılır. Bu sebeple uygulamalar için ayrı realm'ler oluşturulması tavsiye edilir.

2. Kurulum

Keycloak kullanabilmek için, 11 ya da daha yeni bir sürüme sahip Java kurulu olmalıdır. Aynı Apereo CAS'da olduğu gibi, tüm Keycloak kodlarını indirmek yerine, ayağa kaldırmak için gerekli minimal kod ve jar'ların olduğu proje (keycloak-19.0.1.zip) indirilir. Daha sonra binary klasörü altındaki (/bin), kc.bat dosyası, aşağıdaki komut ile çalıştırılır.

- Development ortamı için; kc.bat start-dev

Burada HTTP bağlantı kullanılır. Önbellek mekanizması devre dışıdır. Alan adı (hostname) ayarlamadan da çalışabilir.

- Production ortamı için; kc.bat start

Burada "secure by default" prensibi takip edilir. Eğer ayar yapmadan direkt bu komut calıştırılırsa, hata verir. Çünkü HTTPS ayarı ve alan adı ayarlaması yapılmalıdır.

İlk çalıştırıldıktan sonra, varsayılan ayarlar değiştirilmediyse http://localhost:8080 adresinden giriş ekranına girilir. İlk kez girişte, keycloak yönetim ekranı için admin kullanıcısı oluşturulmalıdır. Bu, hem kullanıcı arayüzünden hem de ortam değişkenleri ile (KEYCLOAK_ADMIN=x KEYCLOAK_ADMIN_PASSWORD=y) yapılabilir. Eğer admin kullanıcısı varsa ve ortam değişkeni de ayarlanmışsa, ayağa kalkma aşamasında hata log'u gözükür ve environment değerler ihmal edilir. Admin oluşturulduktan sonra diğer kullanıcılar, kullanıcı arayüzünden ya da keadm.bat dosyasından oluşturulabilir. Admin kullanıcısı ve normal kullanıcılar oluşturulduktan sonra giriş adresleri farklılık göstermektedir.

Admin kullanıcısı için http://localhost:8080/admin/master/console/ adresinden, Myrealm realm'inde oluşturulmuş kullanıcılar için http://localhost:8080/realms/myrealm/account/ adresinden erişim sağlanır.

3. Paket Yapısı

- /bin altında bat ve sh uzantılı dosyalar ve uygulamayı ayağa kaldırma, kullanıcı oluşturma gibi dosyalar bulunur.
- /conf altındaki dosyalar ile, keycloak ayarları yapılır.
- /data altında keycloak tarafından oluşturulmuş bilgiler saklanır.
- /providers altında keycloak'a dışarıdan eklenmek istenen java paketleri yer alır.
- /themes altında kendimize ait temalar geliştiririz.

4. Keycloak Konfigürasyonu

Keycloak konfigürasyonu için 4 yol vardır. Önem sırası üstten aşağıya gider. Yani üstte tanımlanan alttakileri ezer.

- Komut satırı parametreleri (Command-line parameters), uygulama ayağa kaldırılırken geçilir. Bu tip parametreler -- =x yapısında yazılır. Örnek kullanımı şu şekildedir.

```
kc.bat start-dev --db-url-host=mykeycloakdb
```

- Ortam değişkenleri (Environment variables), KC_=x yapısında yazılır. Örnek kullanımı şu şekildedir.

```
export KC_DB_URL_HOST=mykeycloakdb
kb.bat start-dev
```

- Kullanıcı tarafından tanımlanan konfigürasyon dosyaları (User-created .conf file). Kullanıcının varsayılan conf dosyası hariç oluşturduğu konfigürasyonlar, conf dosyasındaki yapıyla aynı şekildedir. key-with-dashes=x yapısında yazılır.

```
db-url-host=mykeycloakdb
```

Ayrıca conf dosyasının ortam değişkenlerini okuması sağlanabilir.

```
db-urlhost=${MY_DB_HOST} ya da varsayılan değer ile db-url-host=${MY_DB_
HOST:mydb}
```

Örnek Kullanımı:

```
kc.bat start-dev --config-file=/conf/my-custom.conf
```

- Varsayılan keycloak.conf (/conf klasörü altında) dosyası üzerinden yapılandırılabilir.

Keycloak tarafından sağlanan tüm konfigürasyonlara ya https://www.keycloak.org/server/all-config sitesi üzerinden ya da kb.bat start-dev –help komutuyla erişilebilir. Bazen Keycloak'ın desteklemediği fakat Quarkus Framework'ün sağladığı davranışı etkinleştirmek için conf altında quarkus.properties oluşturulur ve ihtiyacımız olan özellikler (property) tanımlanır. https://quarkus.io/guides/all-config ile bu konfigürasyonlara ulaşılabilir. Eğer bir özellik hem quarkus.properties de hem de keycloak.conf'da varsa, keyclok'taki çalışır, çünkü üstünlük ondadır.

5. Keycloak İstemcilerinin Kayıt Edilmesi

Eklenecek istemcilerin ait olacağı realm'e gidilir ve clients tabından yeni bir istemci yaratılır. Keycloak'ın, Apereo CAS gibi kendine ait protokolü yoktur. İstemciler ile olan iletişimini OpenID Connect ya da SAML üzerinden yapar. Burada gerekli bilgiler ile istemci oluşturulur.

Örnek olarak, Spring uygulamasının (geliştirme ortamının 8081 portunda çalışsın) OpenID Connect ile Keycloak üzerinden kimlik doğrulaması yapılmak istenirse, valid redirect URI kısmına http://localhost:8081/* eklenir. Oluşturulan istemci, curl kullanılarak password grant type ile test edilebilir. Bu istek sonucunda, access token elde edilmesi beklenir.

```
curl -X POST -d "client_id=myclient&username=&password= &grant_type=password" -H
"Content-Type: application/x-www-form-urlencoded" http://localhost:8080/realms/
myrealm/protocol/openid-connect/token
```

6. Spring Framework Entegrasyonu

Spring Security'nin Keycloak ile entegre edilebilmesi için Keycloak istemcisi olduğunu belirten bir kütüphane eklemelidir. (org.keycloak:keycloak-spring-boot-starter)

```
keycloak.auth-server-url=http://localhost:8080
keycloak.realm=myrealm
keycloak.resource=myclient
keycloak.public-client=true
```

Kullanıcı, Spring uygulamasındaki herhangi bir uca (endpoint) tıkladığında, Spring Security tarafından sağlanan sso/login ucuna yönlendirilir. Bu uç, KeycloakSpringbootConfigResolver tarafından ele alınır ve Keycloak'a authorization code akışı ile gönderilir. Örnek yönlendirme url'i şu şekildedir:

http://localhost:8080/realms/myrealm/protocol/openidconnect/auth?response_type=code&client_id=myclient&redirect_uri=http%3A%2F%2Flocalhost%3A8082%2Fsso%2Flogin&state=3ca6b64b-f06a-443a-b509- 1bed4991781e&login=true&scope=openid

Kullanıcı, kendisine ait kullanıcı adı ve şifre bilgisini girdikten ve keycloak da bu bilgileri doğruladıktan sonra, authorization code'unu alarak uygulamaya gider ve burada authorization code – access token değişimi yapılarak, kullanıcı girişi sağlanmış olur.

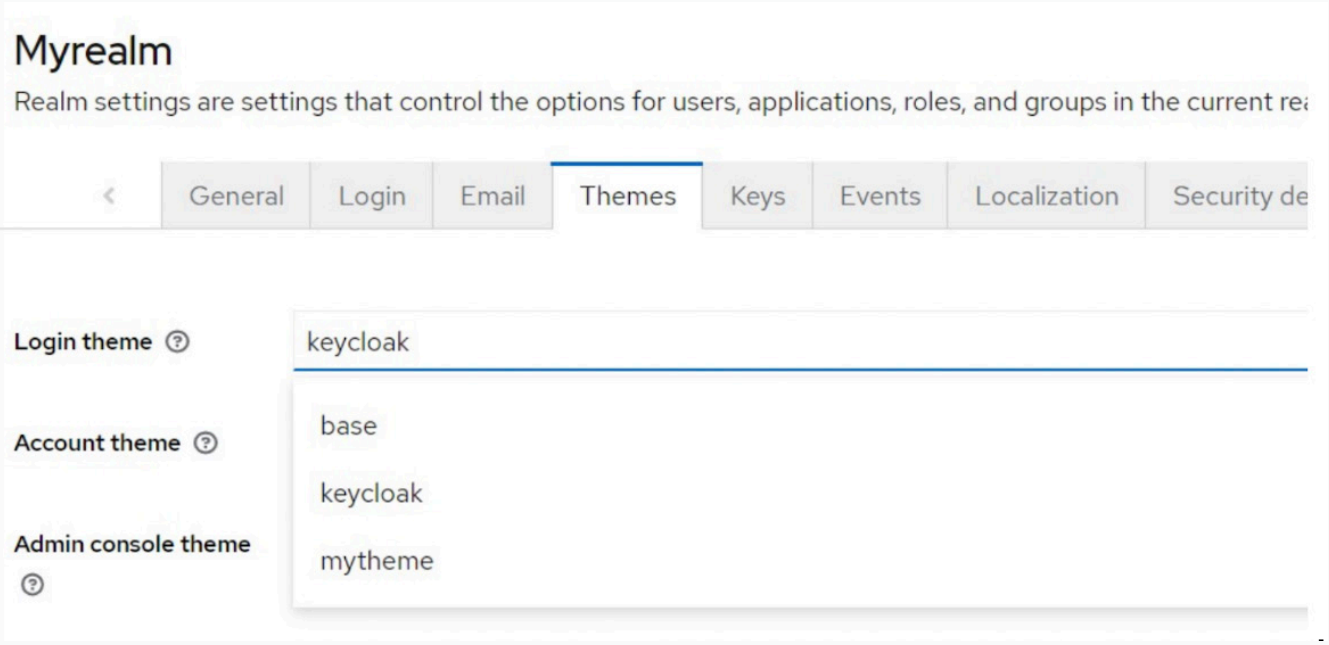
7. Özel Ekran Teması Geliştirme

Keycloak ekranları 6 farklı tipe ayrılır ve her tip, tek başına özelleştirilebilir. Bu tipler; account, admin console, emails, login forms ve welcome page'dir. Welcome hariç hepsi, admin konsolu üzerinden konfigüre edilir. Welcome page de komut satırı parametreleri ile konfigüre edilir. (kc.bat start-dev --spi-theme-welcome-theme=custom-theme)

Bir temayı sıfırdan yazmak yerine, varolan Keycloak temalarını genişletmek (extend) mantıklıdır.

Kendi temamızı oluşturabilmek için themes klasörü altında yeni bir klasör oluşturulur (örnek olarak mytheme. Tema isimlerinde _ dahil hiçbir özel karakter ve sayı olmamalıdır.). Bu klasör altında da, hangi tip ekranın configure edilmesi istenirse, ona özel klasör oluşturulur. Bu klasör isimleri; account, admin, common, email, login ya da welcome isimlerinden birisi olmalıdır.

Şekil 3'de keycloak yönetim ekranında realms settings altında mytheme isminde tema gözükür ve yeni temayı kullanacak şekilde değiştirilebilir.



Şekil 3. Keycloak Tema Seçim Ekranı

Her tip ekran için, o tipin klasörü altında theme.properties oluşturulur. Burada tema icin konfigürasyonlar yapılır. Mesela yeni oluşturulacak tema, keycloak'ın hangi temasını extend edecek ve hangi ortak kaynakları import edecek ayarı için şunlar kullanılır.

```
parent=base
import=common/keycloak
```

Parent ayarında base yerine keycloak kullanıldığında, ekrandaki değişim gözlenebilir. Özel tema için CSS eklemek için, tema ekran tipi klasörü altında /resources/css klasörleri yaratılır ve orada yeni bir css dosyası oluşturulur (mesela styles.css). Daha sonra tema ekran tipinin theme.properties dosyasına styles=css/styles.css ayarı geçilir. Yeni temaya, parent olarak kullanılan temadaki CSS"ler de geçilebilir. Birden fazla CSS dosyası geçebilmek için aralarında boşluk bırakılır.

```
styles=web_modules/@fontawesome/fontawesome-free/css/icons/all.css
web_modules/@patternfly/react-core/dist/styles/base.css
web_modules/@patternfly/reactcore/dist/styles/app.css
node_modules/patternfly/dist/css/patternfly.min.css
node_modules/patternfly/dist/css/patternfly-additions.min.css css/login.css css/
styles.css
```

En sondaki CSS dosyası, kendisinden önceki tanımlanmış dosyaları ezer. Bu yüzden, yeni oluşturulan temaya ait CSS dosyası, en sonda tanımlanmalıdır.

Yeni tema için JavaScript eklemek için, tema ekran tipi klasörü altında /resources/js klasörleri yaratılır ve orada yeni js dosyası oluşturulur (mesela script.js). Daha sonra tema ekran tipinin theme.properties dosyasına scripts=js/script.js ayarı geçilir. Yeni temaya resim eklemek için, tema ekran tipi klasörü altında /resources/img klasörleri yaratılır ve resimler buraya atılır (mesela image.jpg). Bu image, CSS içerisinde url('../img/image.jpg') şeklinde ya da FTL içerisinde src="${url.resourcesPath}/img/image.jpg" şeklinde kullanılır.

Ekranlardaki metinler, message bundle'lardan yüklenir. Eğer yeni yaratılan tema başka bir temayı genişletiyorsa (extend), base temadaki mesajlar da alınır. Base temadaki mesajları ezmek için, tema ekran tipi klasörü altında, /messages/messages_en.properties dosyası oluşturulur. Keycloak, varsayılan olarak ingilizce dilini desteklemektedir. Çoklu dil desteği (internationalization) yapabilmek için, oluşturulan dosyalara locale bilgisi eklenir (messages_<locale>.properties). Daha sonra theme.properties'e locale ayarları geçilir.

```
locales=en,tr
```

Kendi ekranlarımızı oluşturmak için tema ekran tipi klasörü altında template dosyaları oluşturulur. Keycloak, HTML yaratmak için Apache Freemaker Template'leri (FTL) kullanır. Ayrıca base temadan gelen şablonlar da override edilebilir. Tüm şablon listesine keycloak kaynak kodundaki themes/base/ klasörü altından erişilebilir.

Bir tema yaratırken, önbelleği devre dışı bırakmak iyi bir pratiktir, bu sayede keycloak'ı restart etmeye gerek kalmadan themes klasörü altındaki değişikleri direkt olarak görmek mümkün olur.

```
kc.bat start-dev --spi-theme-static-max-age=-1 --spi-theme-cache-themes=false
--spi-theme-cachetemplates=false
```

8. Varsayılan Kodları Ezme (Override)

Keycloak, kendisinin desteklemediği fakat ihtiyaç duyulan şeylerin yazılabilmesi için SPI'lar (Service Provider Interface) sunar. Örnek olarak Keycloak; kerberos, password, OTP gibi farklı kimlik doğrulama mekanizmaları sağlar. Fakat ihtiyaç karşılanmıyorsa, yeni tip mekanizmalar yazabilmek için Authentication SPI da sunar. Authentication SPI kullanarak, varolan akışa yeni bir davranış eklenebilir ya da tamamen yeniden kodlanabilir.

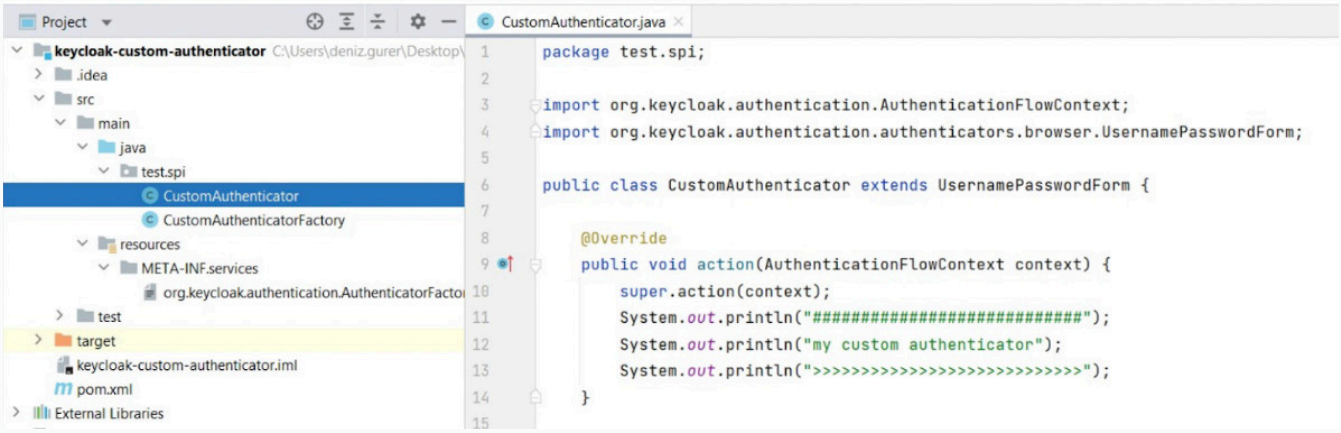
Yazılan özel davranışlar ve provider'lar, keycloak'a dışarıdan jar olarak sağlanır. Uygulama ayağa kalkarken Keycloak, classpath'i tarar ve tüm mevcut provider'ları seçer. Bu yüzden, dışarıdan sağlanan jar'daki META-INF/services klasöründe, sağlanacak servis arayüzünün adını taşıyan bir dosya oluşturulmalıdır. Bu servis klasörü, Keycloak tarafından providerları taramak ve kendi sistemine yüklemek için kullanılır. Keycloak'ta

halihazırda hangi tip servisler var sorusu icin keycloak ekranında "provider info" ekranına bakılabilir. http://localhost:8080/admin/master/console/#/master/providers adresinden kontrol edilebilir.

SPI sütunu altındakiler ve sağdakiler de o SPI için mevcut provider'lardır.

SPI dokümantasyonunda, kullanıcı kimlik doğrulaması için gerekli aksiyonları tanımlayacak şey org.keycloak.authentication.AuthenticatorFactory'dir, dolayısıyla servis klasörü altında bu isimde bir dosya olacaktır. Bu dosya da, oluşturulan özel factory sınıfın paket adı da dahil adını içerir.

```
# SPI class implementation
test.spi.CustomAuthenticatorFactory
```



Şekil 4. Özel SPI Tasarımı

Bu oluşturulan özel provider'ların kullanılabilmesi için, öncelikle jar'ı oluşturulur, daha sonra bu jar, keycloak içerisindeki providers/ klasörü altında kopyalanır. Bu jar keycloak'a eklendikten sonra, uygulama ayağa kaldırılır ve özel provider'ın keycloak'a gelip gelmediği "provider info" ekranında kontrol edilebilir.

Bu provider'ın kullanılabilmesi için, ilgili realm'deki authentication tabından create flow -> isim verilir -> add execution'dan yeni oluşturulan seçilir. Oluşturulduktan sonra sağa tıklanır ve bind flow ile aktifleştirilir. Daha sonra giriş sırasında, yazılan SPI'nın logları kontrol edilir.

9. Canlı (Production) Ortama Hazırlık

Daha hızlı ayağa kalkma ve daha iyi hafıza tüketimi için Keycloak'ı optimize etme yolları vardır. Keycloak, ayağa kaldırılırken start veya start-dev komutlarından biri kullanılır. Bu komutlarda, Keycloak arka planda build komutu çalıştırır. Bu komut, optimize şekilde çalışma elde etmek için çeşitli optimizasyonlar çalıştırır, fakat bu komutun çalışması bir süre alır. Keycloak'ın ayağa kalkma aşamasında derleme esnasında geçen zamanı önlemek için, CI/CD pipeline'da önce, derleme bir süreç olarak çalıştırılabilir. Derlemeden sonra keycloak, --optimized parametresi ile varsayılan davranışı olmadan başlatılır. --optimized parametresi, keycloak'ın önceden derlenmiş olduğunu söyler. Yani, optimize edilmiş keycloak paketi kullanılacaktır. Bu sayede, keycloak'ın ayağa kalkma aşamasında build komutu çalıştırması önlenir.

```
kc.bat start --optimized
```

Sonuç ve Öneriler

Bu çalışmada, yazılım projelerinin önemli bir parçası olan kullanıcı girişi ve yetkilendirilmesi teknolojilerinden Apereo CAS, Keycloak yazılımları ele alınmıştır. İlgili teknolojilerin birbirlerine göre avantajları ve dezavantajları karşılaştırılması aşağıda sunulmuştur.

Apereo CAS Kullanımının Avantajları

Apereo CAS, içerisinde barındırdığı 200'den fazla modül sayesinde, akla gelebilecek ve ihtiyaç duyulabilecek en geniş entegrasyon kümesine sahip enterprise seviyede bir kimlik doğrulama yazılımıdır. Neredeyse sıfır geliştirme ve sadece konfigürasyon ile canlı ortama hazır bir kimlik doğrulama ortaya çıkarılabilir.

Apereo CAS Kullanımının Dezavantajları

- Öğrenme eğrisi yüksektir. Eğer daha önce kurumda CAS tecrübesi olan birisi yoksa, bunu kullanmayı seçmek, geliştirme süreçlerini yavaşlatır.
- Dokümantasyonu ortalamanın çok altındadır. İhtiyaç duyulan bilgiye erişim sınırlıdır.
- Topluluk desteği zayıftır. Stackoverflow, gitter.im gibi soru-cevap platformlarında dönüşler çok azdır. (Örnek olarak, bu yazının yazıldığı tarihte stackoverflow üzerinden son 1 ayda 5 CAS sorusu sorulmuş ve bu soruların hiçbirine cevap gelmemiştir. Bkz: https://stackoverflow.com/tags/cas/topusers) Kullanıcılar arası iletişim, az da olsa eski usül mail grupları üzerinden ilerlemektedir.
- Built-in yönetim ekranına sahip değildir. cas-management-overlay isminde başka bir proje ile, ekranlar üzerinden bazı CAS sunucu ayarlamaları yapılabilir. Fakat bu, çok ilkel bir araçtır, kısıtlı özellikler sunmaktadır.

Keycloak Kullanımının Avantajları

- İlk defa kullanılacaksa, hızlıca PoC (Proof of Concept) yapılabilir. Kurulumu ve entegrasyonu Apereo CAS'a göre çok kolaydır.
- Built-in yönetim ekranına sahip ve ekran üzerinden hızlı işlemler yapılabilir.
- Topluluklarda, sorulan sorulara göre cevaplanma oranı, Apereo CAS'a göre daha iyi. Bkz: https://stackoverflow.com/tags/keycloak/topusers

Keycloak Kullanımının Dezavantajları

Kullandığı authentication yöntemleri ya da entegrasyonlar, Apereo CAS'a göre daha azdır. Fakat SPI kullanarak yeni davranışlar eklenme şansı hala vardır. Apereo CAS'da olduğu gibi hazır modülleri kullanmak yerine, kendi kodumuzu yazmamız gereklidir.

Kaynakça

1. https://github.com/apereo/cas
2. https://github.com/apereo/cas-management
3. https://github.com/apereo/cas-overlay-template
4. https://apereo.github.io/cas/6.6.x/index.html
5. https://fawnoos.com/blog/
6. https://github.com/keycloak/keycloak
7. https://www.keycloak.org/guides
8. https://github.com/Identicum/keycloak-custom-authenticator
9. https://apereo.github.io/cas/6.6.x/installation/Troubleshooting-Guide.html#pkix-path-buildingfailed
10. https://apereo.github.io/cas/development/authentication/Configuring-SSO-Cookie.html#samesite-attribute
11. https://apereo.github.io/cas/6.6.x/ticketing/Redis-Ticket-Registry.html
12. https://www.keycloak.org/server/configuration-production

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgem@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)

yte.bilgem.tubitak.gov.tr